IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION PAPERS

OF

MANUEL NEDBAL

FOR

MANAGING COMPUTER PROGRAM CONFIGURATION DATA

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to the field of data processing systems. More particularly, this invention relates to the management of computer program configuration data, such as, for example, Windows Registry data, within data processing systems.

Description of the Prior Art

It is known to associate configuration data with computer programs. As an example, when a computer program is first installed a user may be allowed to specify certain options as to how that computer program is to be configured. One example would be the language to be used for the display of messages or instructions, parameters specifying the location of different components of the computer program product and the like. Within the current range of Windows operating system programs, this configuration data is normally stored within the Windows Registry. The structure and content of the Windows Registry can be complex and problems within this data can lead to unreliability of the associated computer programs. Furthermore, it is difficult to reliably and efficiently centrally manage such configuration data installed on computer networks and the like. If System Administrators wish to check if the configuration of the computers on their network is correct and depending on the result of this check want to apply a certain configuration, then they may have little choice but to make this check and change for each separate computer and each configuration setting.

SUMMARY OF THE INVENTION

Viewed from one aspect the present invention provides a computer program product for controlling a computer to validate program configuration data, said computer program product comprising:

comparing code operable to compare an XML data representation of said program configuration data with data defining valid program configuration data;

wherein, if said XML data representation does match said data defining valid program configuration data, then triggering code is operable to trigger a valid program configuration response.

The invention recognises and exploits that program configuration data generally has a regular and ordered structure and well defined ranges of valid configuration parameters such that the program configuration data can be well represented as XML data. Once the configuration data is represented as XML data, then existing tools and resources that are provided for the validation of XML data may be applied to the representation of the configuration data as XML data in order to validate that configuration data. This provides a low overhead, flexible and reliable technique for managing program configuration data.

It will be appreciated that the program configuration data could take a wide variety of forms. As previously discussed, the program configuration data could be registry data associated with the operating system of the computer concerned. Alternatively, program initialisation data, such as "ini" data could form the configuration data. Another possibility is that the configuration data may in its native form be XML data. Generally any configuration store whose data can be mapped into XML data can be used.

It will be appreciated that when the configuration data is not XML data in its native form, then it is necessary to provide a mapping mechanism for mapping the program configuration data into an XML data representation of that program configuration data and vice versa. Thus, live existing configuration data may be mapped and compared to establish validity and for other management reasons.

The data defining valid program configuration data could take a variety of forms, but particularly preferred embodiments use XSD data or DTD data for this purpose. The tools and skills for validating XML data using these schemas are already provided and can be exploited in this way.

Whilst it would be possible to provide special purpose mechanisms for comparing the XML data representation with the validity data, preferred embodiments use an XML parser, which will typically already be provided within a computer system for validating XML data from other sources and for other reasons. This advantageously reduces the overhead required to exploit the above techniques.

In preferred embodiments of the invention the mapping between the program configuration data and the XML data representation takes place via a DOM data representation, which is a common way of holding and manipulating XML data in the computer's main memory. Making this mapping via this intermediate form allows the tools, mechanisms and the skills normally provided for working with DOM data to be reused for the purpose of managing, and in particular editing and adjusting program configuration data, whilst keeping within an overall arrangement that allows rigorous validation of the results of such editing or modification.

It will be appreciated that the validation of the configuration data could take place at a variety of different points within an overall system. In some embodiments it may be desirable to validate the configuration data within a program configuration managing computer before such configuration data is sent out to an associated managed computer. Alternatively and/or additionally the managed computer may operate to validate configuration data that it receives in accordance with the above described techniques.

Viewed from another aspect the present invention provides a computer program product for providing program configuration data for a computer, said computer program product comprising:

receiving code operable to receive an XML data representation of said program configuration data at said computer;

mapping code operable to map between said XML data representation and said program configuration data; and

configuration code operable to at least one of apply said program configuration data to said computer and retrieve said program configuration data from said computer.

This aspect of the invention principally relates to a managed computer which receives configuration data generically specified as XML data and then maps this to the native form for that managed computer before applying that configuration data to the managed computer. Mapping in the opposite direction is also enabled. The XML data can be validated upon receipt and/or before it is sent. This provides a robust and

flexible mechanism for distributing and validating program configuration data and allowing changes to that data as appropriate.

Other aspects of the invention provide methods and apparatus operating in accordance with the above described techniques.

The above, and other objects, features and advantages of this invention will be apparent from the following detailed description of illustrative embodiments which is to be read in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 schematically illustrates the software architecture at a target computer for processing autonomously and remotely initiated target processes;

Figure 2 schematically illustrates the software architecture at a target computer supporting different types of target process;

Figure 3 schematically illustrates the software architecture at a target computer for deploying software configuration data;

Figure 4 schematically illustrates the software architecture at a target computer for retrieving configuration data;

Figure 5 is a flow diagram schematically illustrating the processing performed by an initiating computer seeking to trigger a target process;

Figure 6 is a flow diagram schematically illustrating the processing performed by an agent process on the target computer for receiving data for initiating target processes;

Figure 7 is a flow diagram schematically illustrating the processing performed by a target process;

Figure 8 is a diagram representing a protocol specification of a data transmission protocol that may be used to trigger execution processes at a remote computer;

Figure 9 is an example XML schema for XML data in accordance with the transmission protocol of Figure 8 that may be used to control execution processes at the destination computer;

Figures 10A and 10B are an example of XML data that may form a request to execute an execution process at a destination computer;

Figure 11 is an example of XML data that may be returned as a response from an execution process such as that initiated in connection with the XML data of Figures 10A and 10B;

Figure 12 is an XML schema for XML data that may be used to trigger an execution process for updating Windows Registry data;

Figure 13 is an XML schema for XML data that may be used to update configuration data within an INI file;

Figure 14 is an XML schema that may be used to validate XML data for updating configuration data held within a DAPI store;

Figures 15A and 15B show an example of XML data that may be used to transmit a configuration update to a destination computer using a selection of different execution processes upon the destination computer to perform the different parts of the configuration update;

Figure 16 is a flow diagram schematically illustrating the processing performed at a source computer to assemble an XML data transmission for triggering an execution process at a destination computer;

Figure 17 is a flow diagram schematically illustrating the processing performed at a destination computer in responding to XML data for triggering an execution process at that destination computer;

Figure 18 schematically represents Windows Registry data that specifies a computer program configuration;

Figure 19 illustrates a representation of DOM data as made accessible in memory corresponding to the configuration data of Figure 18;

Figure 20 is an XML data representation of the configuration data of Figure 18;

Figure 21 is a first representation of an XSD schema corresponding to the XML data of Figure 20;

Figure 22 is a second representation of an XSD schema corresponding to the XML data of Figure 20;

Figure 23 is a flow diagram schematically illustrating the mapping of configuration data to XML data and then the validation of this XML data;

Figure 24 schematically illustrates an application program checking its own configuration data;

Figure 25 schematically illustrates the editing of configuration data and its validation by an application program;

Figure 26 schematically illustrates the technique of Figure 25 followed by the transfer of the validated XML data to a managed computer and the application of that XML data to the managed computer;

Figure 27 illustrates a modification of the technique of Figure 26 in which the managed computer also checks the received XML data again to validate it before it is applied; and

Figure 28 is a schematic diagram representing a general purpose computer of the type that may be used to implement the above described techniques.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figure 1 schematically illustrates a software architecture which may be used within a target computer operating in accordance with one example of the herein described techniques. An agent process 300 receives operation specifying data in the form of XML data and optionally validating XML schema data. The XML schema data (XSD) may be applied to the received XML data to check that it meets the required form before it is used. The agent process then parses the XML data using an XML parser. The XML parser may already be provided within the target computer for use with other applications and processes that deal with XML data. Alternatively, if required, a specific XML parser could be provided solely for the use of the agent process 300.

When the XML data has been validated and parsed, different complex data types specified within the XML data will be recognised. Each of these complex data types corresponds to a different target process (execution process) on the target computer (destination computer). Accordingly, the different complex data types may be matched with the target processes that are available at the target computer. These different target processes are indicated by the software devices 302, 304 and 306. These target devices can take a wide variety of different forms and should not be considered to be limited to the specific examples discussed herein. Each target device 302, 304 and 306 is responsible for a corresponding area of processing (tasks) 308, 310 and 312. These areas of processing could again take a wide variety of forms and should not be limited to the specific examples discussed herein.

In the example of Figure 1, the XML data received is parsed and the complex data type corresponding to the software device 306 is identified. This device 306 is then triggered to execute and is passed appropriate parameter data that was also embedded within the corresponding complex data type. The device 306 performs the required processing upon its processing area 312 and generates corresponding result data. The device 306 then packs this result data back into XML data which is returned to the agent process 300 and then in turn to the initiating computer.

It will be appreciated that the target processes/execution processes/devices are substantially independent and separate from the agent process. This enables the ready

extension of the number of target processes available. Use of the complex data type to identify the particular target process allows a structured and robust architecture to be formed and yet provides enough flexibility to allow extension, user customisation and a wide variety of parameter data to be utilised.

Figure 2 is similar to Figure 1 except more specific examples of target processes are illustrated in the form of an API device 314, an install device 316 and an event device 318. These different devices 314, 316 and 318 correspond to different execution processes which may be remotely triggered using the XML data protocol. More particularly, the API device 314 may be used to trigger a specified Win32 API function call. The install device 316 may be used to make configuration data changes and install associated computer files as part of software installation/updating activity. The event device 318 may be used to trigger processing to perform specific events or monitor the target computer to alert the initiating computer upon the occurrence of particular events. Figure 2 gives an indication of the flexibility and extensibility of the agent architecture associated with the current technique.

Figure 3 schematically illustrates a yet more specific example of the architecture on the target computer relating to the deployment of configuration data in association with remote software management. In this example the target processes 320, 322 and 324 respectively relate to software for mapping between parameter data embedded within the corresponding XML complex data types which trigger those processes and the Windows Registry, an INI file and a DAPI store respectively (a mapping to another form of database that holds configuration data is also possible). In this example, the XML data received by the agent process 300 contains three complex data types respectively corresponding to the different target processes 320, 322 and 324. The agent process 300 parses the XML data after validating it with the XSD data which forms the XML schema. The parsing of the XML data extracts the corresponding identifiers of the complex data types which can then be mapped to the available target processes 320, 322 and 324. If a complex data type is encountered which does not have a corresponding available target process on the target computer, then data indicating this may be returned to the initiating computer embedded within the XML protocol of a returned message. In this example, the target process 320 writes to make changes to the Windows Registry 326 as specified by parameter data associated with the complex data type for the device

320. The device 322 makes specified changes or writes a new INI file as directed by its associated parameter data. The device 324 makes changes to an associated DAPI store 328 in accordance with a binary passed as parameter data for the complex data type of the device 324.

Figure 4 is closely related to Figure 3. In the example of Figure 4, the XML data received from the initiating computer corresponds to a request by the initiating computer for each of the devices 320, 322 and 324 to read their corresponding configuration data store and return the contents thereof to the initiating computer as a response embedded within XML data. The software mechanisms within the devices 320, 322 and 324 that map between XML data and configuration data may be substantially reused to provide this data transfer in the opposite direction to that illustrated in Figure 3. The request for configuration data may be passed to the various devices by way of a requesting XML data transmission in which each of the complex data types corresponding to the different devices is present and identified but with empty parameter data. The corresponding devices 320, 322 and 324 respectively receive their portion of the XML data with its empty parameter data and interpret this as a request to fill that empty parameter data from the data held within the configuration data store to which they control mapping. In this way, detailed configuration data may be returned to the initiating computer.

Figure 5 is a flow diagram schematically illustrating the processing performed by the initiating computer. At step 330 the initiating computer waits for a user input or automatic event that indicates that a remote operation should be triggered. Step 332 identifies the target process which is to be triggered. Step 334 forms the parameter data that is to be associated with the processing to be triggered in the target process. It may be that in some examples no parameter data is required. Alternatively, other target processes may require highly complex and extensive parameter data.

At step 336, the XML data representing the identified target process and parameter data is assembled. Examples of this XML data will be described in more detail later in the specific examples illustrated. In general terms, the XML data has a complex data type corresponding to the target process to be triggered and the parameter data is specified as data within that corresponding complex data type.

At step 338 the XML data is sent to the target computer. At step 340 the initiating computer waits for response XML data to be returned from the target computer. At step 342 received response XML data is analysed. It will be appreciated that steps 340 and 342 are optional and may not be required in many circumstances.

Figure 6 is a flow diagram schematically illustrating the processing performed by the agent process 300. At step 344 the agent process waits for XML data to be received from the initiating computer. At step 346 received XML data is validated against XML schema data. This XML schema data may be sent to the agent process 300 from the initiating computer at the same time as the XML data or alternatively may be already present within the agent process 300. If the XML data does not pass its validation test, then processing returns to step 344. If the XML data is successfully validated, then processing proceeds to step 348 at which the XML data is parsed and the target process identifiers are read as the complex data types within the XML data. These target process identifiers are then matched with the available target processes within the target computer and the one or more appropriate target processes are triggered to operate at step 350. Step 350 also passes to the triggered target processes any associated parameter data within their respective complex data types contained in the received XML data.

At step 352 the agent waits for a response to be received from the target process. At step 354 the response received is packed into XML data to be returned to the initiating computer at step 356. It will be appreciated that steps 354 and 356 are optional.

Figure 7 is a flow diagram schematically illustrating the processing that is performed by a target process. At step 358 the target process waits to receive an indication from the managing agent process 300 that it should initiate operation. When operation is initiated, then step 360 operates to receive any parameter data that is being passed to the target process. It may be that in some examples no parameter data is required. At step 362 the target process performs its associated processing operation on the target computer. The processing operation performed may take place on the target computer itself or may take place using a processing source available to the target computer but physically separate therefrom. At step 364 any optional result parameter data is returned to the agent process 300.

Figure 8 gives example details of an XML protocol which may be used to trigger the execution of an execution process at a destination computer from a source computer. It should be particularly noted within Figure 8 that in this example the field <CustomActions> corresponds to a complex data type within the XML, data which in turn maps to an execution process to be triggered. Beneath this complex data type resides various optional parameter data associated with the processing being triggered.

Figure 9 graphically illustrates XML schema data corresponding to the protocol of Figure 8. This XML schema data may be used by a target computer/destination computer to validate received XML data. It will be seen that the parameter data resides beneath the custom data type within the data structure.

Figures 10A and 10B show an example of XML data in accordance with one example of the techniques described herein. It will be seen that this XML data corresponds to the XML schema of Figure 9 and is an example of input data for Figure 2. Within this XML data are embedded a plurality of complex data types each corresponding to a CustomAction and having associated parameter data (except the initial ControlData, which may not be necessary and will not be described any further.) The XML data of figures 10A and 10B is a CustomActionRequest specifying processing operations to be performed by the destination computer in accordance with the parameter data specified.

Figure 11 illustrates example XML data, which can be used as input data for Figure 2 and corresponds to Figures 9, 10A and 10B, but in this case the CustomAction being specified is a Response from the destination computer whereby the destination computer will return parameter data stored at or accessible to the destination computer. This requirement for the parameter data to be returned may be indicated by providing empty parameter fields within the XML data of Figure 11.

Figures 12, 13 and 14 respectively illustrate XML schemas corresponding to the portions of XML data to be sent to a target computer/destination computer to manage the configuration of that computer broadly in accordance with the arrangement illustrated in Figures 3 and 4. The respective XML schemas of Figures 12, 13 and 14 apply to the

portions of the XML data being different complex data types which will be concatenated together to form the XML data transferred in order respectively to control Windows Registry configuration, INI file configuration and DAPI store configuration.

Figures 15A and 15B show an example of XML data for Figures 3 and 4 for updating the configuration of a target computer/destination computer in accordance with software installation/management using custom data types corresponding to those of Figures 12, 13 and 14. Associated with each of these custom data types is embedded parameter data which will be used by the corresponding target processes/execution processes within the target computer/destination computer.

Figure 16 is a flow diagram schematically illustrating the processing performed at a source computer in assembling XML data in accordance with the protocol described herein. At step 366 the source computer waits for user input or an automatic event indicating that a remote operation should be triggered. At step 368, the remote operation to be triggered is identified and XML data is assembled including a complex data type corresponding to that remote process to be triggered. It may be that more than one remote process is to be triggered and that respective complex data types may be concatenated within the XML data generated.

At step 370, any required parameter data is added to the XML data to specify the remote processing required by the target process/execute process. At step 372, the XML data that has been generated and assembled is transmitted to the destination computer.

Figure 17 is a flow diagram schematically illustrating the processing performed at a destination computer at which execution processes may be remotely triggered. At step 374, the destination computer waits for XML data in accordance with the above described protocol to be received. At step 376, received XML data is parsed to identify the complex data types contained therein. A standard XML parser provided within the destination computer for other purposes may be reused for this parsing, or alternatively a specific XML parser may be provided for the agent process 300. At step 378 the complex data types identified are matched to corresponding execution processes at the destination computer. If a matching execution process is not present, then an appropriate message indicating this may be returned to the source computer by embedding this

within returned XML data. At step 380 the execution process indicated by the matched complex data type is triggered including passing any associated parameter data to the execution process concerned.

Figure 18 schematically represents a portion of the program configuration data associated with an application program that is stored within the Windows Registry of a computer using the Windows operating system provided by Microsoft Corporation. This data is illustrated as being accessed and viewed via the Registry editor tool that is conventionally provided. It will be seen that the configuration data specifies parameters such as the version of the computer program installed, devices associated with that installation, the language to be used in particular circumstances with associated messages for display in response to particular events that may occur during execution and the like. The specification of computer program configuration data within the Windows Registry in this way is well known within the field of computer programming and will not be described herein any further.

Figure 19 schematically illustrates the configuration data of Figure 18 which has been mapped into DOM data and is being viewed by an associated viewing tool. The mapping of the configuration data into DOM data can be achieved in a wide variety of different ways. One example, is to map keys within the Windows Registry data to complex data types within the DOM data. Values within the Windows Registry can be mapped to simple types within the DOM data. In order to deal efficiently with keys and values within the Windows Registry that can occur any number of times the mapping mechanism may parse the Windows Registry to identify such keys and types and then associate attributes with the different instances of the keys and types encountered. This helps to provide a more efficient and compact DOM representation of the configuration data which will ultimately result in a more compact and efficient XML representation, with associated XSD validating data.

A table illustrating one example of how Windows Registry configuration data may be mapped in accordance with the present technique is given below.

Registry Data	XSD Data	XML Data	Comments
Values correspond to XML elements			

"valuename"="stringvalue"	<xs:element name="valuename" type="xs:string"/>	<valuename>stringvalue </valuename>	*REG_MULTI_SZ strings can be mapped into lists of XML strings and the other way round.*
"valuename"=dword:dwordvalue	<xs:element name="valuename" type="xs:unsignedLong" />	<valuename>dwordvalue </valuename>	*Conversions between hexadecimal and decimal forms have to be taken into consideration here.*
"valuename"=hex:hexvalue	<xs:element name="valuename" type="xs:hexBinary"/>	<valuename>hexvalue</ valuename>	
Keys correspond to XML complex types			
[keyname]	<xs:complexType name="keyname"><xs:all> ... *All subkeys and values* ... </xs:all></xs:complexType>	<keyname> ... *All subkeys and values* ...</keyname>	*<xs:all> means that this type's sequence of its members does not matter (in contrast to <xs:sequence>)*
Enumeration of Values, which may occur any number of times			
"valuename id=valueid"=" somevalue "	<xs:element name="valuename" id="valueid" type="sometype"> somevalue </xs:element>	<valuename id="valueid" type="sometype"> somevalue </valuename>	*Changing the format of the registry value name simplifies the XSD validation. The name contains the additional information "valueid", which distinguishes each value from the others.*
Enumeration of Keys, which may occur any number of times			
[keyname id="keyid"]	<xs:complexType name="keyname" id="keyid"><xs:all> ... *All subkeys and values* ... </xs:all></xs:complexType>	<keyname id="keyid"> ... *All subkeys and values* ...</keyname>	*Similar to the Enumeration of Values, except that any kind and number of subkeys and –values are allowed.*

Known commercially available tools may be used to map from the DOM data as represented in Figure 19 to corresponding XML data as illustrated in Figure 20. The highly flexible and robust nature of XML as a representation of data will be appreciated from Figure 20. Furthermore, a comparison of the Windows Registry data of Figure 1 stored in its heirarchical structure will be appreciated to map well to the hierarchical data representation that is associated with XML. The recognition and exploitation of the fact that configuration data has a form and structure that is well suited to representation by XML data is an important feature of the current technique and enables many existing tools and resources provided for general XML data

manipulation and validation to be reused for the manipulation, management and validation of program configuration data once this is transformed into the form of XML data.

As well as the tools for mapping configuration data into DOM data and XML data, the validation technique described herein also requires associated validation data in the form of XSD data against which XML data may be checked. This XSD data will normally be generated by the provider of the computer program product which is having its configuration managed in accordance with this technique or it can be generated by programs knowing the configuration data of another program. An efficient way to generate this XSD data is to start with a known valid set of Windows Registry configuration data for the computer program concerned. Once this has been mapped into XML data using the above described mapping technique, a tool such as XMLSpy may be applied to that example XML data representation of the configuration to produce an associated XSD validation template. Figure 21 illustrates one view of the XSD data that may be generated from the XML data previously discussed using such an automated tool. Such an automated tool typically will not provide the most elegant and compact XSD data corresponding to the XML representation of the configuration data. Accordingly, once the tool has produced the XSD data shown in accordance with Figure 21, it is proposed that this XSD data would then be examined and hand edited by a programmer familiar with the application concerned. Such hand editing will typically provide a more compact and realistic XSD representation of the configuration data as well as allowing ranges of valid configuration parameters to be specified in a manner generalised from the particular configuration parameters that may be picked up from the example configuration that was converted using the automated tool. This hand editing of XSD data is a general technique used by those familiar with the technical field and will not be described further.

Figure 22 illustrates an example of XSD data that may be associated with the previously described configuration data and has been generated by hand editing of an automatically provided XSD representation of that XML data.

It will be appreciated that the technique of the present invention is not restricted to the mechanisms for generating associated XSD data as described above nor the particular form of configuration validating data represented by XSD data. However, these techniques are strongly preferred as they do enable the reuse of overhead, resources and skills that are generally already provided.

Figure 23 is a flow diagram illustrating the validation of program configuration data. Windows Registry data 2 is mapped into DOM data 4 by a mapping function 6. This mapping function may operate using the example mappings described above, or alternative mappings that may be possible. The DOM data 4 is streamed into XML data 8 by step 10. This streaming may also be referred to as serialisation. The XML data 8 together with previously generated and associated XSD data 12 is then provided to an XML parser 14 where the XML data 8 is validated against the XSD data to produce a validation result 16. The XML parsers and validation mechanisms are typically already provided within internet browsers and the like which are installed on computers for reasons other than the validation of configuration data.

Figure 24 is a flow diagram illustrating an application program validating its configuration. An application program 18 is provided with an associated set of XSD data 20 by the provider of that application program 18. This XSD data 20 can be viewed as a template for valid configuration data. The provider of the application program 18 will use their knowledge and expertise relating to that application program 18 in order to provide a generic and robust set of XSD data.

The Windows Registry data 22 corresponding to the particular installation concerned is read and mapped into DOM data 24 before being serialised into corresponding XML data 26 which represents the configuration data (Windows Registry data 22). A call may be then made to an XML parser 28 which validates the XML data of the particular installation against the XSD data 20 provided in association with the application program in order to generate a validation result 30. It will be appreciated that the validation result 30 may be used to trigger either an invalid configuration response, such as generation of an appropriate error message, or

a valid configuration response, such, for example, as starting execution of the associated application program.

Figure 25 is a flow diagram schematically illustrating the editing of configuration data using the present techniques. It will be seen that a number of the steps in Figure 25 correspond to those in Figure 24 and these will not be described again. Compared with Figure 24, the DOM data 24 is made available to other applications which may modify that DOM data 24 to customise it or edit it in other ways. An editing application 32 may be used to hand edit the DOM data 24. Alternatively and/or additionally, XML data 34 may be de-serialised and appended to or inserted within the DOM data 24 in a manner to extend the configuration data. Once the editing of the DOM data 24 has been completed, this edited DOM data 24 is serialised into XML data 36 which is then validated against the corresponding XSD data 20. If the validation step is passed at step 38, then the modified configuration data represented by the XML data 36 is mapped back into DOM data and then Windows Registry data 40 for use within the computer concerned to replace the original Windows Registry data 42.

Figure 26 illustrates the process of Figure 25 but in this case the successfully validated XML data is transmitted to another computer (a managed computer) at step 44. The received XML data 46 is de-serialised within the managed computer to form DOM data 48 which is then in turn mapped into Windows Registry data 50 for controlling the configuration of a different instance of the application program concerned that is installed within the managed computer. It will be appreciated that the editing and validation of the configuration data which occurred in the first portion of Figure 26 is carried out by a configuration managing computer, such as a computer operated by a System Administrator, and once this edited program configuration has been successfully validated it is automatically distributed to associated managed computers and applied to those managed computers. In the example of Figure 26, the managed computer does not itself recheck the validity of the configuration data which it receives. Instead it receives this data in the form of an XML data representation of its configuration which it maps into the required native configuration data and then applies this native configuration data.

Figure 27 illustrates a modification of the technique of Figure 26. After the XML data 44 representing the configuration has been transmitted (transferred) to the managed computer, the managed computer uses its own copy of the application program 52 concerned to read XSD data associated with the configuration data such that this may be validated by the managed computer itself before being applied. In this particular instance, the XML data representation of the configuration is validated both within the configuration managing computer and the managed computer. It would be possible, but probably less efficient, to only validate the data within the managed computer.

Figure 28 schematically illustrates a general purpose computer 200 of the type that may be used to implement the above described techniques. The general purpose computer 200 includes a central processing unit 202, a random access memory 204, a read only memory 206, a network interface card 208, a hard disk drive 210, a display driver 212 and monitor 214 and a user input/output circuit 216 with a keyboard 218 and mouse 220 all connected via a common bus 222. In operation the central processing unit 202 will execute computer program instructions that may be stored in one or more of the random access memory 204, the read only memory 206 and the hard disk drive 210 or dynamically downloaded via the network interface card 208. The results of the processing performed may be displayed to a user via the display driver 212 and the monitor 214. User inputs for controlling the operation of the general purpose computer 200 may be received via the user input output circuit 216 from the keyboard 218 or the mouse 220. It will be appreciated that the computer program could be written in a variety of different computer languages. The computer program may be stored and distributed on a recording medium or dynamically downloaded to the general purpose computer 200. When operating under control of an appropriate computer program, the general purpose computer 200 can perform the above described techniques and can be considered to form an apparatus for performing the above described technique. The architecture of the general purpose computer 200 could vary considerably and Figure 28 is only one example.

Although illustrative embodiments of the invention have been described in detail herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to those precise embodiments, and that various changes and

modifications can be effected therein by one skilled in the art without departing from the scope and spirit of the invention as defined by the appended claims.